An, Meng-Ai

From: Courtenay, St.John
Sent: Wednesday, January 21, 2004 11:50 AM
To: An, Meng-Ai
Subject: Results of search for 09/408,149

Meng,

I spent at least 3 hours searching 09/408,149 and examining the specification and several possible references.

I found a particular reference (U.S. Pat. 5,995,745) that is prior art directed to the same problem (i.e., a real time operating system); however my reference discloses a main real time operating system that has both preemptive and non preemptive modes of operation. The reference also discloses a second general purpose (non real time) operating system running as a task within the main real time operating system.

Preemption is defined when a currently executing task or process involuntarily gives up control to the processor (usually through timer interrupts, or a real time interrupt that triggers a context switch where control is passed to the next waiting process).

In contrast, non preemption is defined when a task gives up the processor voluntarily, e.g., after the task has run to completion. This definition is set forth in a related reference, U.S. patent 6,507,861, col. 2, line 18.

U.S. Pat. 5,995,745 discloses a scheduler that has both preemptive and non preemptive modes of operation, these modes are mutually exclusive with respect to time - i.e., they obviously cannot occur simultaneously.

The non preemptive and preemptive modes of operation in the '745 patent are disclosed in col. 4, line 53: "Tasks give up the processor voluntarily, OR are preempted by a higher priority task ..."

Also, a recent case of mine, 10/226,106 (first office action outstanding), contains non patent references that are material to the instant case, although I am not sure if the dates are good. The case is in Franconia.

If this were my case, I would apply new grounds of rejection using the '745 patent for at least the independent claims.

In addition, U.S. patent 5,721,922 is a possible second refernce for this case.

Thanks,
St. John

An Introduction to RTLinux

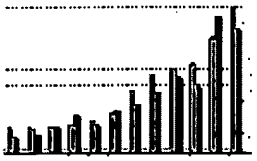
Victor Yodaiken, New Mexico Institute of Technology (Oct. 7, 1997)

RT-Linux is an operating system in which a small real-time kernel coexists with the Posix-like Linux kernel. The intention is to make use of the sophisticated services and highly optimized average case behavior of a standard time-shared computer system while still permitting real-time functions to operate in a predictable and low-latency environment. At one time, real-time operating systems were primitive, simple executives that did little more than offer a library of routines. But real-time applications now routinely require access to TCP/IP, graphical display and windowing systems, file and data base systems, and other services that are neither primitive nor simple. One solution is to add these non-real-time services to the basic real-time kernel, as has been done for the venerable VxWorks and, in a different way, for the QNX microkernel. A second solution is to modify a standard kernel to make it completely pre-emptable. This is the approach taken by the developers of RT-IX (Modcomp). RT-Linux is based on a third path in which a simple real-time executive runs a non-real-time kernel as its lowest priority task, using a virtual machine layer to make the standard kernel fully pre-emptable.

In RT-Linux, all interrupts are initially handled by the Real-Time kernel and are passed to the Linux task only when there are no real-time tasks to run. To minimize changes in the Linux kernel, it is provided with an emulation of the interrupt control hardware. Thus, when Linux has "disabled" interrupts, the emulation software will queue interrupts that have been passed on by the Real-Time kernel. Real-time and Linux user tasks communicate through lock-free queues and shared memory in the current system. From the application programmers point of view, the queues look very much like standard UNIX character devices, accessed via POSIX

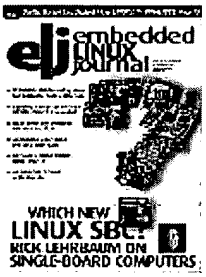
- Is embedding Windows cheaper than Linux?
- Consumer Electronics I Foundation
- Where's the free beer?
- ELC Platform Spec
- Motorola adopts Linux f phones

Embedded Linux mark report:



Embedded Linux Quick Reference Guides
- Intro & overview
- Embedded distributions
- Real-time Linux
- Embedded GUI/Windo
- Embedded Java + Linu
- Linux-based cool devic
- Linux-based PDAs
- Little Linux systems
- Single board computer
- System-on-Chip proce

Read ELJonline here:



Got news? submit
. . . articles? submit
. . . products? submit

Popular discussions
- Article talkbacks
- Linux on PDAs
- Technical questions
- Real-time Linux
- GPL issues, discussio
- Embedded Java+Linu
- Little Linux systems
- General discussion

More discussions here

Other cool websites
- NewsForge NewsVac
- Linux Today
- Linux Daily News
- Linux Weekly News
- Linux Journal

read/write/open/ioctl system calls. Shared memory is currently accessed via the POSIX mmap calls. RT-Linux relies on Linux for booting, most device drivers, networking, file-systems, Linux process control, and for the loadable kernel modules which are used to make the real-time system extensible and easily modifiable. Real-time applications consist of real-time tasks that are incorporated in loadable kernel modules and Linux/UNIX processes that take care of data-logging, display, network access, and any other functions that are not constrained by worst case behavior.

In practice, the RT-Linux approach has proven to be very successful. Worst case interrupt latency on a 486/33Mhz PC measures well under 30 microseconds, close to the hardware limit. Many applications appear to benefit from a synergy between the real-time system and the average case optimized standard operating system. For example, data-acquisition applications are usually composed a simple polling or interrupt driven real-time task that pipes data through a queue to a Linux process that takes care of logging and display. In such cases, the I/O buffering and aggregation performed by Linux provides a high level of average case performance while the real-time task meets strict worst-case limited deadlines.

RT-Linux is both spartan and extensible in accord with two, somewhat contradictory design premises. The first design premise is that the truly time constrained components of a real-time application are not compatible with dynamic resource allocation, complex synchronization, or anything else that introduces either hard to bound delays or significant overhead. The most widely used configuration of RT-Linux offers primitive tasks with only statically allocated memory, no address space protection, a simple fixed priority scheduler with no protection against impossible schedules, hard interrupt disabling and shared memory as the only synchronization primitives between real-time tasks, and a limited range of operations on the FIFO queues connecting real-time tasks to Linux processes. The environment is not really as austere as all that, however, because the rich collection of services provided by the non-real-time kernel are easily accessed by Linux user tasks. Non-real-time components of applications migrate to Linux. One area where we hope to be able to make particular use of this paradigm is in QOS, where it seems reasonable to factor applications into hard real-time components that collect or distribute time sensitive data, and Linux processes or threads that monitor data rates, negotiate for process time, and adjust algorithms.

The second design premise is that little is known about

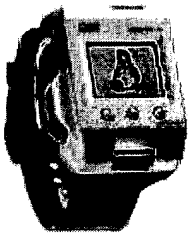
- Linux Magazine
- Linux kernel downloads
- General Linux links
- SiliconPenguin.com
- Linux International
- Embedded Linux newsgroup

Use our news feed: info

how real-time systems should be organized and the operating system should allow for great flexibility in such things as the characteristics of real-time tasks, communication, and synchronization. The kernel has been designed with replaceable modules wherever practical and the spartan environment described in the previous paragraph is easily "improved" (or "cluttered", depending on one's point of view). There are alternative scheduling modules, some contributed by the user community, to allow for EDF and rate-monotonic scheduling of tasks. There is a "semaphore module" and there is active development of a richer set of system services. Linux makes it possible for these services to be offered by loadable kernel modules so that the fundamental operation of the real-time kernel is run-time (although not real-time) reconfigurable. It is possible to develop a set of tasks under RT-Linux, test a system using a EDF schedule, unload the EDF scheduling module, load a rate monotonic scheduling module, and continue the test. It should eventually be possible to use a memory protected process model, to test different implementations of IPCs, and to otherwise tinker with the system until the right mix of services is found.

. . . For further information

| about us | contact us | terms of use | home | search | directory | news | articles | polls | forum | events | jobs | products | links | sponsors |



RTLinux -- a hard real-time Linux add-on

RTLinux is a hard real-time operating system that runs Linux as its lowest priority execution thread. The Linux thread is made completely preemptible so that realtime threads and interrupt handlers are never delayed by non-real-time operations. Real-time threads in RTLinux can communicate with Linux processes via shared memory or a file-like interface, so real-time applications can make use of all the powerful, non-real-time services of Linux. For example, it is easy to write a Perl script that displays data in Xwindows, responds to commands delivered over a network, and collects data from a real-time task.

RTLinux supports real-time interrupt handlers and realtime periodic tasks with interrupt latencies and scheduling jitter close to hardware limits. Worst case interrupt latency on a modest, reasonably configured x86 PC is under 15 microseconds from the moment the hardware interrupt is asserted. Better hardware configurations produce better timings. (This is a worst case scenario, not typical time. Time is measured from the moment the hardware asserts, not when the operating system scheduler runs.)

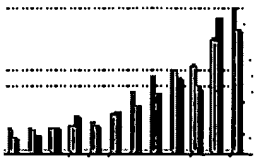
Realtime tasks in RTLinux can communicate with Linux processes via shared memory or a file-like interface. Thus, real-time applications can make use of all the powerful, non-real-time services of Linux, including:

- Networking
- Graphics
- Windowing systems
- Data analysis packages
- Linux device drivers, and
- Standard POSIX API

It is quite simple, for example, to create a Perl script that displays data in Xwindows, responds to commands

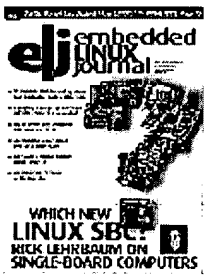
• Is embedding Windows cheaper than Linux?
• Consumer Electronics I Foundation
• Where's the free beer?
• ELC Platform Spec
• Motorola adopts Linux f phones

Embedded Linux mark report:



Embedded Linux Quick Reference Guides
• Intro & overview
• Embedded distributions
• Real-time Linux
• Embedded GUI/Windo
• Embedded Java + Linu
• Linux-based cool devic
• Linux-based PDAs
• Little Linux systems
• Single board computers
• System-on-Chip proces

Read ELJonline here:



Got news? submit
. . . articles? submit
. . . products? submit

Popular discussions
• Article talkbacks
• Linux on PDAs
• Technical questions
• Real-time Linux
• GPL issues, discussion
• Embedded Java+Linux
• Little Linux systems
• General discussion
More discussions here

Other cool websites
• NewsForge NewsVac
• Linux Today
• Linux Daily News
• Linux Weekly News
• Linux Journal

delivered over a network, and collects data from a real-time task.

Three versions of RTLinux are available via ftp:

- RTLinux V1 is an exceptionally stable system. It is in use both in delivered products and in laboratory environments. V1 RTLinux provides a simple API for starting, stopping, and scheduling realtime tasks.
- RTLinux V2 offers a simplified version of the POSIX pthreads API, and will be compliant with the POSIX "Minimal Real-time" standard (in which the realtime component is considered a single, multithreaded POSIX process). The aim of the V2 kernel is to offer near-POSIX with no sacrifice of speed and simplicity, while proving hooks for add-ons for POSIX full compatibility. V2, released in November 1999, is an SMP capable x86 system.
- RTLinux V3 Beta, released January 3, 2000, adds PowerPC support.

RTLinux, originally developed at the New Mexico Institute of Technology, is an open-source product. RTLinux-specific components are released under the GPL, and Linux components released under the standard Linux license. The source code is freely distributed. Non-GPL versions of the RTL components are available from FSMLabs.

RTLinux.com discusses commercial support and development.

Other articles about RTLinux:

- Victor Yodaiken on the past, present, and future of RTLinux -- a comprehensive interview with the chief maintainer of RTLinux
- The miniRTL project -- a miniature real-time Linux -- a brief introduction to miniRTL, which fits a full RTLinux-based standalone system within the storage capacity of a single floppy
- Real Time Linux for Embedded Systems in the Internet Era -- a copy of the presentation given by Victor Yodaiken at the Embedded Linux Expo & Conference, June 2000
- An Introduction to RTLinux -- one of the early whitepapers by Victor Yodaiken describing RTLinux, its architecture, and how it is used
- The RTLinux Manifesto -- whitepaper by Victor Yodaiken which discusses the requirements of hard

- Linux Magazine
- Linux kernel downloads
- General Linux links
- SiliconPenguin.com
- Linux International
- Embedded Linux newsgroup

Use our news feed:
info

real-time systems and how RTLinux meets those needs
- FSMLabs releases RTLinux version 3.0-pre8
- New RTLinux release (V2.2) extends POSIX compliance

. . . For further information

L1: Entry 4 of 4 File: USPT Apr 21, 1998

DOCUMENT-IDENTIFIER: US 5742825 A
TITLE: Operating system for office machines

Abstract Text (1):
A multitasking, graphical windowing operating system for an office machine supporting real time processing, including a method for supporting real time processing in the operating system. The operating system includes a kernel having a non-preemptive scheduler for scheduling windowing applications and a preemptive scheduler for scheduling real time applications. The kernel includes a messaging subsystem for communication among process in the system including both the real time and windowing applications, and further includes an event semaphore maintained by the kernel to manage scheduling. The method includes grouping windowing applications together for scheduling purposes and scheduling windowing applications non-preemptively. The method further includes determining priority of real time applications and scheduling real time applications preemptively with the windowing applications based on priority.

Brief Summary Text (12):
Though beneficial from the standpoint of compatibility, modifying an existing operating system can give rise to difficult issues in scheduling. The Windows Operating System, for example, does not support real time processing because it is based on a non-preemptive scheduling scheme. An operating system that is based on a non-preemptive scheduling scheme must be modified to include preemptive processing to support real time processing. At the same time, the original non-preemptive scheme must be kept in tact to maintain compatibility with existing applications.

Brief Summary Text (13):
In modifying a non-preemptive operating system to support real time processing, reentrancy problems must be addressed because many of the software modules in a non-preemptive system are non-reentrant.

Brief Summary Text (16):
The invention provides a method and system for supporting real time processing in a graphical windowing operating system. In one embodiment, a method according to the invention includes providing a graphical windowing operating system having a non-preemptive scheduler, a preemptive scheduler, and a messaging subsystem for allowing the schedulers to communicate with processes. The method further includes grouping windowing applications together as one scheduable process called a system process and scheduling windowing applications non-preemptively with the non-preemptive scheduler. The method further includes scheduling the system process and real time applications preemptively with the preemptive scheduler. Preemptive scheduling includes determining whether the currently running process can be preempted, scheduling a process that is ready to be scheduled based on the type and priority of the currently running process, and preempting the currently running process by performing a context switch.

Brief Summary Text (18):
In addition to the methods outlined above, the invention provides a system for supporting real time applications in an operating system for an office machine. According to one embodiment of the system, the system includes a kernel having a messaging subsystem, a non-preemptive scheduler for scheduling windowing applications, a preemptive scheduler for scheduling real time applications, and an event semaphore. The messaging subsystem enables real time and windowing applications to communicate with each other and with processes in an operating system using messages.

Brief Summary Text (19):
The messaging subsystem is also used in scheduling. Real time and windowing applications may transition states based upon whether they are posting or getting a message. The messaging subsystem is in communication with the event semaphore for controlling scheduling by communicating the state of a process to the proper scheduler. The non-preemptive scheduler schedules windowing applications non-preemptively, while the preemptive scheduler schedules real time applications preemptively based upon the priority of applications ready to be scheduled.

Brief Summary Text (21):
The design of a system according to the invention may vary. For instance, the kernel may be implemented in two pieces: one supporting non-preemptive scheduling, and a second supporting preemptive scheduling. The messaging subsystem may use separate message queues for windowing and real time applications to maintain independence between windowing processes operating in the foreground and real time processes operating in the background. Many other variations are possible and will become apparent in the detailed description to follow.

Brief Summary Text (22):
The methods and systems according to the invention can be used to help build a real time, graphical windowing operating system for an office machine. This operating system can be designed to support non-preemptive scheduling for windowing applications and preemptive scheduling for real time applications. Both forms of scheduling may be accommodated while avoiding reentrancy problems. The invention may be used build an operating system for an office machine that retains compatibility with existing windowing applications as well as an existing development environment.

Drawing Description Text (5):
FIG. 4 is a process state diagram of a non-preemptive scheduler from the perspective of an operating system kernel.

Drawing Description Text (6):
FIG. 5 is a process state diagram of a non-preemptive scheduler from the perspective of the user module in an embodiment of the invention.

Detailed Description Text (13):
In general, the kernel 26 provides scheduling, memory management, and IPC communication for two kinds of processes: non-preemptive processes that run in the foreground, and preemptive processes that run in the background. The non-preemptive processes may include Microsoft Windows applications and office machine I/O applications using the display device while the preemptive processes may include background processes, not using the display device. The kernel 26 includes services to create and delete these processes, and services to communicate among them. In this embodiment, processes communicate using the standard Windows Inter-Process Communication messaging mechanism. In addition to the Windows IPC messages, the kernel 26 provides additional communication services, including semaphores and event signals. Processes and dynamic linked libraries may manage resource contention with semaphores and may use signals to notify each other of events. For memory management, the kernel 26 implements global pools of standard size memory blocks in order to reduce fragmentation and to speed up dynamic allocation and deallocation of memory.

Detailed Description Text (15):
FIG. 2 is a more detailed diagram of an operating system according to an embodiment of the invention. In this particular embodiment, the kernel 26 is in two pieces: the office machine operating system (OMOS) kernel 40, and the Windows kernel 42. The kernel 26 may also be implemented as a single module as depicted in FIG. 1. This particular implementation, however, uses the Windows kernel 42 and standard Windows modules to provide part of the operating system functionality, namely to support non-preemptive processing of graphical windowing applications. The OMOS kernel provides the added functionality to support preemptive processing for real time applications. The Windows modules 44 correspond to the graphical services module of FIG. 1 and include a graphics device interface (GDI) and a user module. Both the GDI and user module are standard Windows modules and have been well understood in the art.

Detailed Description Text (20):
Foreground applications, including office machine I/O applications and Windows applications, are scheduled non-preemptively. The communication among the non-preemptive applications and the

operating system is more complicated because some office machine I/O applications must talk to the office machine OS kernel yet also have access to graphical services. For example, an application that acts as a user interface for a fax machine may need access to the office machine OS kernel 40 for scheduling and communication purposes, and may need access to the display through the GDI and user module 44. Office machine I/O applications, therefore, may communicate with the office machine OS kernel 40, which then communicates with the Windows modules 44.

Detailed Description Text (22):
FIG. 3 is a block diagram illustrating communication among software processes in the operating system 20. The communication among processes is performed through messages. Each process has a main message loop in which it retrieves messages sent to it from other processes and responds to them. All inputs from I/O devices such as a keyboard or a mouse are sent as messages. Messaging services for Windows applications are implemented in a messaging subsystem in the user module 80. The messaging services, however, need not be located in the user module 80. They can also be implemented as part of the kernel 26 of the operating system 20. This particular implementation includes messaging services in the user module 80 from the Windows operating system, and additionally, includes a messaging subsystem in the office machine OS kernel that is similar to the Windows messaging services. The advantage to having a separate messaging subsystem implemented in the office machine OS kernel 40 is that it enables office machine background processes to operate independently from the user module 80, which is involved in non-preemptive scheduling of foreground processes. Background processes can operate entirely in the background if necessary.

Detailed Description Text (27):
The user module 80 controls the state of the Windows processes 82 communicating with it. A process basically has three states: blocked, ready, and running. As described above, a Windows process 82 is blocked during a GetMessage() call. In response to a blocking call such as GetMessage(), the user module 80 blocks a process. The user module 80 controls the process "ready" and "running" states by deciding when to place a process in a run queue. To communicate the state of a Windows process 82 to the Windows kernel 42, the user module controls an event semaphore within the Windows Kernel through a semaphore call 88. Using this event semaphore, the Windows kernel 42 then performs non-preemptive scheduling of Windows processes 82.

Detailed Description Text (57):
Having now described communication among processes in the operating system, it is now possible to describe process scheduling. The operating system has a non-preemptive scheduler for scheduling foreground processes such as Windows applications, and a preemptive scheduler for scheduling real-time processes such as office machine background processes. In this embodiment, the Windows kernel 42 supports non-preemptive scheduling while the office machine OS kernel 40 provides preemptive scheduling. First the non-preemptive scheduler of the Windows Kernel will be described, and then two implementations of a scheduler in the operating system will be described. Though the scheduler of the operating system in this embodiment is implemented in two separate kernels, it should be understood that the scheduler could be implemented in a single kernel. Other variations in scheduler implementation exist, but in each, the scheduler provides preemptive scheduling to support real time applications.

Detailed Description Text (58):
The Windows kernel 42 provides non-preemptive scheduling. The Windows kernel has a priority based scheduler. The manner in which priority is assigned to the non-preemptive processes is not critical to the invention and may be implemented in various ways. Basically, processes are executed in the order in which they are listed in a linked list of processes. Processes are executed in a round robin fashion. The scheduler within the Windows kernel 42 only attempts to schedule a process when a current process makes a yield call to the Windows kernel 42. In a standard Windows configuration, the user module 80 makes a yield call on certain messaging calls like GetMessage. Because the non-preemptive processes make frequent yield calls in the process of checking the message queue, each process does not have to wait too long to be executed.

Detailed Description Text (59):
FIG. 4 is a process state diagram of the non-preemptive scheduler from the perspective of the Windows kernel 42. The non-preemptive Windows processes 82 can be in one of three states: blocked 100, ready 102, and running 104. This diagram illustrates transitions between states and the events that trigger the transitions. The user module 80 controls transitions among

states by making the appropriate calls to the Windows kernel 42. To control transitions to the blocked and ready states 100, 102, the user module 80 calls WaitEvent 106 and PostEvent 108, two standard APIs for the Windows kernel 42. PostEvent 108 and WaitEvent 106 manipulate an event semaphore in the Windows kernel 42. The Windows kernel 42 can them determine the state of processes by examining a value in the event semaphore. To control transitions from the running (104) to the ready (102) state, the user module 80 makes a yield call 110 to the Windows kernel 42. The Windows kernel 42 schedules all ready processes in a non-preemptive manner. When scheduled (112), the process transitions from the ready (102) to the running state 104.

Detailed Description Text (60):
FIG. 5 is a process state diagram of the non-preemptive scheduler from the perspective of a process interacting with the functions provided by the user module 80. When viewed together, FIGS. 4 and 5 show the complete picture of state transitions of a Windows process 82. When a process makes a GetMessage call 114, the user module 80 makes the WaitEvent call (106) to the Windows kernel 42 and the process is blocked (100). When a new message arrives (116) in the message queue for the process, the user module 80 makes a PostEvent call 108 and the process is ready (102) for scheduling. Once ready for scheduling, the Windows kernel 42 schedules the process (118), and it begins to run (104). When a running process calls any Windows message API (120), the user module 80 makes a yield call 110, and the process transfers from running to a ready state 104, 102. In this manner, Windows and foreground processes 82, 84 may be scheduled non-preemptively in the operating system.

Detailed Description Text (68):
In preempting a currently executing process, the operating system must avoid reentrancy problems. In a system that is non-preemptive such as the Windows Operating System, much of the code is non-reentrant. If a process is non-reentrant, then it may not be preempted and then reentered while in a preempted state. Windows processes cannot be preempted and then reentered. Due to the Windows display paradigm, it is not possible to timeslice among Windows processes using the display. The problem arises because these Windows processes each call the same standard Windows modules like the user module which are not re-enterable. Background processes, however, do not make direct calls to the user module or use the display. As such, background process can be preemptively timesliced with foreground processes with minimal reentrancy problems.

Detailed Description Text (70):
The preemptive scheduling outlined above may be implemented in a number of ways. In the scheduler embodiments described below, the scheduler is implemented using the Windows kernel 42 and the office machine OS kernel 40. Conceptually, the scheduler can be thought of as residing in a single operating system kernel 26 as depicted in FIG. 1. For the purposes of describing the operating system, however, it is helpful to describe the detailed implementation of the scheduler, including both non-preemptive and preemptive scheduling, at a lower level.

Detailed Description Text (83):
FIGS. 8 and 9 are process state diagrams for a high end office machine embodiment. FIG. 8 illustrates the process state diagram of a foreground process. Foreground processes are treated as one system process. FIG. 9 illustrates the process state diagram of an office machine background process. These background processes are real time processes that are scheduled preemptively with the system process. Though similar to the process state diagrams of FIGS. 4 and 7, the process state diagrams are more complex because foreground and background processes are scheduled differently. Since a process may be in different states in the preemptive and non-preemptive scheduler, the diagrams show the state of the process from the perspective of each scheduler for each state transition. The process state diagram of a foreground process, depicted in FIG. 8, illustrates the transitions among states of a foreground process in both the preemptive and non-preemptive scheduler. The abbreviations, NPS and PS, stand for non-preemptive scheduler and preemptive scheduler, respectively.

Detailed Description Text (84):
The state of the foreground process is the same in each scheduler for each transition except when the foreground process is preempted by the preemptive scheduler (146). In this case, the foreground process transitions to the ready state in the non-preemptive scheduler while it is preempted by the preemptive scheduler (150). Once the preemptive scheduler again schedules the preempted process (148), it is in the running state in both the preemptive and non-preemptive schedulers (152). The preemptive scheduler schedules the preempted process for execution on the next time slice.

Detailed Description Text (85):
Aside from the case where a foreground process is preempted, it is scheduled in all other respects as described with reference to FIGS. 4 and 5. Unless preempted, a foreground process transitions among running (152), blocked (154), and ready (156) states just as in FIG. 4. A transition from a running state 152 to a blocked state 154 occurs on a WaitEvent call 158 to the Windows kernel. A transition from the blocked state 154 to the ready state 156 occurs on a PostEvent call 160 to the Windows kernel. A running process transitions to the ready state 156 on a yield call 162, and a ready process gets scheduled by the non-preemptive scheduler 164. The non-preemptive scheduling is thus very similar to that depicted in FIG. 4.

Detailed Description Text (86):
The process state diagram of a background process, depicted in FIG. 9, illustrates the transitions among states of background process in both the preemptive and non-preemptive scheduler. The state of a background process is the same in each scheduler for each transition except when the background process calls a non-reentrant module in the operating system (170). In this case, the process is blocked in the preemptive scheduler, and it is ready for scheduling by the non-preemptive scheduler (172). Once scheduled by the non-preemptive scheduler (174), the background process transitions to the running state in both schedulers (176).

Detailed Description Text (87):
While a background process always communicates with the office machine OS kernel 40, it can be scheduled by either the non-preemptive scheduler or the preemptive scheduler. On a WaitEvent call 78 to the office machine OS kernel 40, the background process transitions from a running (176) to a blocked state 180 in both schedulers. On a PostEvent call 182 to the office machine OS kernel 40, the background process transitions from a blocked (180) to a ready state 184. Once in a ready state, the background process may be scheduled non-preemptively by the non-preemptive scheduler or preemptively by the preemptive scheduler (186). The background process transitions from a running state 176 in both schedulers to a ready state in both schedulers on either a yield call to the preemptive scheduler or if preempted by the preemptive scheduler (188).

Detailed Description Text (103):
While the operating system is described in detail with reference to particular embodiments, it should be understood that the invention can be implemented in a variety of ways. For example, instead of using separate kernels, the preemptive and non-preemptive schedulers may be implemented in a single kernel as shown in FIG. 1. The messaging services in the system can be implemented in a single messaging subsystem as opposed to using messaging services in the user module and a messaging subsystem in the office machine OS kernel.

Other Reference Publication (3):
Morgan, Kevin D, "The RTOS Difference" BYTE, Aug. 1992 p. 161 vol. 17, No. 8.

CLAIMS:

1. A method for supporting real time processing in a graphical windowing operating system for an office machine, the graphical windowing operating system having a non-preemptive scheduler for scheduling windowing applications, a preemptive scheduler for preemptively scheduling the windowing applications and real time applications, a first messaging subsystem for processing messages for the windowing applications, and a second messaging subsystem for processing messages for the real time applications, the office machine having a processor and memory, the method comprising:

separately controlling communication among the windowing applications with the first messaging subsystem, including posting messages and getting messages for each of the windowing applications in message queues allocated for the windowing applications;

controlling whether the windowing applications are in a blocked or ready state for scheduling in response to message calls to the first messaging subsystem from the windowing applications;

allocating message queues corresponding to the real time applications with the second messaging subsystem;

separately processing messages for the real time applications with the second messaging subsystem including posting and getting messages for each of the real time applications in the message queues allocated for the real time applications;

controlling whether the real time applications are in a blocked or ready state for scheduling in response to message calls to the second messaging subsystem from the real time applications;

allocating message queues for office machine foreground applications in the first messaging subsystem;

receiving message calls from office machine foreground applications in the second messaging subsystem to enable the office machine foreground applications to communicate with the real time applications;

determining whether any of the message calls received in the second messaging subsystem correspond to the message queues allocated for the windowing applications and the office machine foreground applications maintained by the first message subsystem;

making message calls on behalf of a first office machine foreground application in response to receiving a first message call in the second messaging subsystem that corresponds to one of the message queues allocated by the first messaging subsystem for either one of the windowing applications or one of the office machine foreground applications;

scheduling the windowing applications non-preemptively in a system process with the non-preemptive scheduler;

scheduling a first real time application and the system process preemptively by the following steps:

transferring control to the preemptive scheduler upon an interrupt generated by an interrupt module;

determining whether a currently running process can be preempted;

identifying whether the currently running process is in the system process;

determining whether the real time application is ready to be scheduled by the preemptive scheduler; and

if the currently running process is in the system process and the real time application is ready to be scheduled, then preempting the currently running process by performing a context switch between the currently running process and the first real time application.

13. An operating system for an office machine comprising:

a first messaging subsystem operable to allocate and maintain message queues for windowing processes and office machine foreground processes, operable to post and get messages from the message queues to allow the windowing processes and the office machine foreground processes to communicate with each other;

a second messaging subsystem in communication with the office machine foreground processes and office machine background processes, the second messaging subsystem operable to allocate and separately maintain message queues for the office machine background processes so that the office machine background processes can operate independently from the windowing processes, and operable to post and get messages for the office machine background processes to allow the office machine foreground and background processes to communicate with each other, the second messaging subsystem in communication with the first messaging subsystem for making message calls to the first messaging subsystem on behalf of the office machine foreground processes in response to receiving a message call in the second messaging subsystem that corresponds to one of the message queues allocated for the windowing processes by the first messaging subsystem to enable the windowing processes and the office machine foreground processes to communicate with each other;

a non-preemptive scheduler in communication with the first messaging subsystem for non-preemptively scheduling the windowing processes and the office machine foreground processes in a system process;

a preemptive scheduler in communication with the first and second messaging subsystems for preemptively scheduling the office machine background processes and the system process, the preemptive scheduler operable to determine priority of the office machine background processes, and operable to preemptively schedule the office machine background processes based on the priority of the office machine background processes; and

an interrupt module in communication with the preemptive scheduler for issuing interrupts to cause the preemptive scheduler to schedule the office machine background processes.

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Wednesday, January 21, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR	
☐	L57	l3 and L44	0
		DB=USPT; PLUR=YES; OP=OR	
☐	L56	5081577.pn.	1
☐	L55	5414848.pn.	1
☐	L54	5469571.pn.	1
☐	L53	5515538.pn.	1
☐	L52	5528513.pn.	1
☐	L51	5721922.pn.	1
☐	L50	5721922.pn.	1
☐	L49	4553202.pn.	1
☐	L48	4903258.pn.	1
☐	L47	4993017.pn.	1
☐	L46	5291614.pn.	1
☐	L45	5721922.pn.	1
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR	
☐	L44	linux adj process	7
☐	L43	l2 and l35	27
☐	L42	l2 and l39	0
☐	L41	l2 and 39	609
☐	L40	multitasking and L39	1
☐	L39	l19 and L38	34
☐	L38	l21 and 36	295
☐	L37	L36 and l3	0
☐	L36	(dos or windows) and L35	116
☐	L35	(linux or unix) and L34	158
☐	L34	microkern$11	381
☐	L33	microkern$1	0
		DB=USPT; PLUR=YES; OP=OR	
☐	L32	microkernel	261
☐	L31	5506988.pn.	1
☐	L30	5535380.pn.	1
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR	
☐	L29	L2 and L28	27

☐	L28	microkern$1 and (linux or unix)	158
☐	L27	linux.ti.	40
☐	L26	L2 and L25	19
☐	L25	multitask$3 same (linux or unix) same window$1	237
☐	L24	L16.ti.	5
☐	L23	L3 same L22	0
☐	L22	multitask$3 same L21	28
☐	L21	linux same process	614
☐	L20	L3 and L19	2
☐	L19	run same (linux or unix) same windows	1745
☐	L18	run adj linux adj under adj windows	0
☐	L17	L3 same L16	0
☐	L16	linux same windows	3022
☐	L15	courtenay and L14	23
☐	L14	(718/107 |718/108).ccls.	667
	DB=USPT; PLUR=YES; OP=OR		
☐	L13	5721922.pn.	1
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
☐	L12	6092095.pn.	2
☐	L11	multitask same L10	0
☐	L10	posix	548
☐	L9	linux and L8	0
☐	L8	L4	47
☐	L7	(linux or unix) same process$2 same L3	0
	DB=USPT; PLUR=YES; OP=OR		
☐	L6	(linux or unix) same process$2 same L3	0
☐	L5	5301326.pn.	1
☐	L4	L2 and L3	47
☐	L3	non$1preemptive	263
☐	L2	(718/100 |718/101 |718/102 |718/103 |718/104 |718/105 |718/106 |718/107 |718/108).ccls.	2620
☐	L1	non$1preemptive adj $5kernel	1

END OF SEARCH HISTORY

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 5721922 A

L50: Entry 1 of 1 File: USPT Feb 24, 1998

US-PAT-NO: 5721922
DOCUMENT-IDENTIFIER: US 5721922 A

TITLE: Embedding a real-time multi-tasking kernel in a non-real-time operating system

DATE-ISSUED: February 24, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Dingwall; Thomas J.	Portland	OR		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Intel Corporation	Santa Clara	CA			02

APPL-NO: 08/ 323044 [PALM]
DATE FILED: October 13, 1994

INT-CL: [06] G06 F 9/46

US-CL-ISSUED: 395/673; 395/677, 395/572, 395/806, 395/826, 395/840, 395/856, 395/868
US-CL-CURRENT: 718/103; 710/20, 710/36, 710/48, 710/52, 710/6, 715/500.1, 718/107

FIELD-OF-SEARCH: 395/650, 395/673, 395/677, 395/872, 395/806, 395/826, 395/840, 395/856, 395/868, 364/280, 364/281.3

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5081577	January 1992	Hatle	395/827
5414848	May 1995	Sandage et al.	395/650
5469571	November 1995	Bunnell	395/700
5515538	May 1996	Kleiman	395/733
5528513	June 1996	Vaitzblit et al.	364/514A

ART-UNIT: 236

PRIMARY-EXAMINER: Toplu; Lucien U.

ATTY-AGENT-FIRM: Blakely, Sokoloff, Taylor & Zafman

ABSTRACT:

The invention provides a method and apparatus for embedding a real-time multi-tasking kernel in a non-real-time operating system. Through encapsulating a real-time kernel into the interrupt handling environment of a non-real-time graphical user interface, such as Windows.RTM., the method of the present invention allows for an entire real-time environment to be supported within the graphical user interface. The scheduler of the real-time kernel supports multiple threads of execution all running at higher priority than the graphical user interface tasks. By using synchronization mechanisms of the graphical user interface, e.g. V.times.D events in enhanced mode Windows.RTM., the real-time tasks are able to make use of system services of the graphical user interface.

36 Claims, 3 Drawing figures

Terms	Documents
5721922.pn.	1

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 4993017 A

L47: Entry 1 of 1 File: USPT Feb 12, 1991

US-PAT-NO: 4993017
DOCUMENT-IDENTIFIER: US 4993017 A

TITLE: Modularly structured ISDN communication system

DATE-ISSUED: February 12, 1991

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bachinger; Gerhard	Sauerlach			DE
Barkmeyer; Thomas	Munich			DE
Kroesa; Ehrard	Groebenzell			DE
Siegmund; Wolfgang	Munich			DE
Werres; Bernhard	Graefelfing			DE

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Siemens Aktiengesellschaft	Berlin and Munich			DE	03

APPL-NO: 07/ 313508 [PALM]
DATE FILED: February 22, 1989

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
DE	3808639	March 15, 1988
DE	3833351	September 30, 1988

INT-CL: [05] H04Q 11/04

US-CL-ISSUED: 370/58.2; 370/60
US-CL-CURRENT: 370/360

FIELD-OF-SEARCH: 370/60, 370/60.1, 370/58.1, 370/58.2, 370/58.3, 370/94.1, 370/94.2, 370/94.3, 370/67, 370/66

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4903258	February 1990	Kuhlmann et al.	370/58.2

OTHER PUBLICATIONS

"ISDN in the Office -HICOM", Technology and Applications of the HICOM ISDN Communication System, Special Issue Telcom Report and Siemens magazine COM, Dec. 1985, ISBN 3-8009-3846-4, pp. 4-111.
"High Performance Microprocessor with Integrated Memory Management", Intel Corporation, Oct. 1985, No. 231630-001.

ART-UNIT: 263

PRIMARY-EXAMINER: Olms; Douglas W.

ASSISTANT-EXAMINER: Chin; Wellington

ATTY-AGENT-FIRM: Hill, Van Santen, Steadman & Simpson

ABSTRACT:

A modularly structured ISDN communication system has a system processor that is administered by a multi-tasking real-time operating system. Since this operating system can only assign system-internal function programs to the system processor (on-line system), a software application module having a time-sharing operating system is provided, so that the same processor can be administered by both operating systems. In order to effect this, one task of the real-time operating system is fashioned as a sluice sub-system SUB in which an allocation switch from the addressing tables of the real-time operating system onto those of the time-sharing operating system occurs. An interrupt transition routine that leads back into the sluice sub-system SUB in which the allocation switch is in turn cancelled is present in the time-sharing operating system for a processing of interrupt requests. Messages can be forwarded from one operating system to the other by use of an intercommunication data segment, so that data from the administration of the one operating system can be transferred into that of the other operating system in a simple manner.

13 Claims, 12 Drawing figures

US-PAT-NO: 5995745
DOCUMENT-IDENTIFIER: US 5995745 A

TITLE: Adding real-time support to general purpose operating systems

DATE-ISSUED: November 30, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Yodaiken; Victor J.	Socorro	NM	87801	

APPL-NO: 08/ 967146 [PALM]
DATE FILED: November 10, 1997

PARENT-CASE:
This application claims the benefit of U.S. provisional application Ser. No. 60/033,743 filed Dec. 23, 1996.

INT-CL: [06] G06 F 9/455

US-CL-ISSUED: 395/500.47; 395/500.43, 709/103, 710/262
US-CL-CURRENT: 703/26; 703/22, 710/262, 718/103

FIELD-OF-SEARCH: 395/500, 395/670, 395/677, 395/733, 395/735, 395/500.43, 395/500.44, 395/500.47, 709/100, 709/102, 709/103, 709/107, 710/260, 710/262

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4553202	November 1985	Trufyn	710/269
☐	4993017	February 1991	Bachinger et al.	370/360
☐	5291614	March 1994	Baker et al.	712/35
☐	5721922	February 1998	Dingwall	709/103

OTHER PUBLICATIONS

H. Lycklama et al., "UNIX Time-Sharing System: The MERT Operating System," The Bell System Technical Journal, vol. 57, No. 6, pp. 2049-2086, Jul.-Aug. 1978.

ART-UNIT: 273

PRIMARY-EXAMINER: Teska; Kevin J.

ASSISTANT-EXAMINER: Frejd; Russell W.

ATTY-AGENT-FIRM: Wilson; Ray G.

ABSTRACT:

A general purpose computer operating system is run using a real time operating system. A real time operating system is provided for running real time tasks. A general purpose operating system is provided as one of the real time tasks. The general purpose operating system is preempted as needed for the real time tasks and is prevented from blocking preemption of the non-real time tasks.

11 Claims, 7 Drawing figures